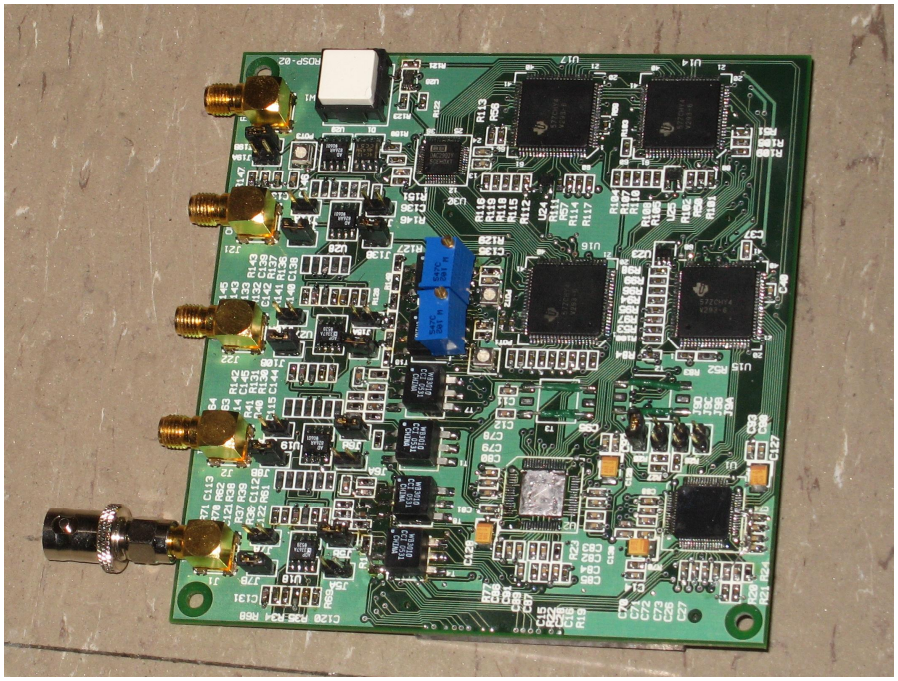
Time Reversal DSP

Technical Description of DSP Signal
Acquisition and Transmission Board



ECE 405
Design III
Jonathan Blixt
David Hinkemeyer
Kurt Pulczinski

https://saturn.ece.ndsu.nodak.edu/ecewiki/index.php/Group_248

Table of Contents

Chapter 1 - Introduction

1.1 Abstract

This document reviews the process taken to develop a daughter card for a Spectrum Digital TMS320C6416 *Digital Signal Processor Starter Kit* (DSK). The purpose of the card is to provide a means for time reversal and processing of signals. It has been designed to allow the researcher to capture two high frequency channels or two composite channels made up of several low frequency signals modulated on separate carriers. Regardless of the type of signal that is sent in, the daughter card will digitize it so that it may be time reversed and processed by the Digital *Signal Processor* or DSP. Once the DSP has finished with the signal, it can then be transmitted back out using the daughter card. The DSK will interface the daughter card using its built-in *External Memory Interface* (EMIF). The daughter card will be connected to an application specific card, which has not yet been created. The additional card will contain filter and amplification blocks.

1.2 Background

Time reversed signal processing is the process of sending a signal over a channel in one direction, receiving that signal over an array of antennas, and then sending the reversed mirrored signal back out of the received array. This signal, in theory, will then take the exact same path back to the source as it took coming to the array, thus converging back at the source. The signal that is returned is an autocorrelated version of the original signal. Time reversal signal processing is an effective way to obtain measured data and improve system performance. Some of the applications that have used this theory are lithotripsy (the blasting of kidney stones), underwater communications, and broadband wireless data transfer.

1.3 Problem

The Department of Electrical and Computer Engineering currently possesses DSKs for processing such signals as desired for time reversal. In order for the DSK to process the signal, the signal needs some sort of interface to the DSK. This creates the need for a daughter card that interconnects to the DSK that will receive the signal, process the signal, and transmit the time-reversed signal. The daughter card will have to provide two channels of data acquisition and transmission capability. A main problem that occurs when doing this type of project is in analyzing the signals. The signals need to be converted from analog to digital in order to facilitate a time reversal. The problem therein is that this acquisition needs to be done at a relatively high frequency. This is because the researcher needs to be able to manipulate a wide range of frequencies.

1.4 Objective

The goal of this senior design project is to design a daughter card that will acquire and transmit two high frequency channels and allow them to be processed on the DSK.

1.5 Potential Issues

- Extremely small packages of components desired
  - Inability to breadboard the components
  - High level of difficulty in hand soldering
  - Potential for board shorts
- Complexity of Printed Circuit Board (PCB)
  - The board must be fabricated out of house
  - High frequency considerations needed
  - Size constraints must be carefully regulated
  - High cost issues
- Interfacing the software with hardware
  - There are many registers that must be set to appropriate values
  - Application specific configuration is needed
- Reliability
- Overall functionality
- Schematic and PCB design
  - To learn and effectively use a design tool like Mentor Graphics and Expedition PCB takes years of practice where the group must learn and use it within weeks

Chapter 2 – Previous Work

2.1 Introduction

There has been much research into this subject, however much of it has been involving lower frequency ranges. Much of this research had been done for military, medical, and communication applications.

2.2 Published Articles

Many various articles have been published on this topic. Most of them can be found on various sites throughout the internet. They discuss many aspects of this topic and many cover the communication theory of how this concept works. The papers that have been used in this project are referenced in the final chapter of this project.

2.3 Previous Senior Design Team

A senior design team has already undertaken this project. The previous group did not attempt as broad of a signal range as is proposed in this design, however, many aspects are similar such as the block diagram and the code. This team succeeded in building a daughter card and in capturing and transmitting a signal. Due to time constraints, testing of the concept of time reversal was limited.

Chapter 3 – Design Specifications

3.1 General Overview

Each of the two channels of the Signal Analysis Daughter Card should be designed to do the following (in order):

1. Acquire an analog signal.
2. Filter out the high frequency content.
3. Convert the analog signal to a digital signal.
4. Buffer the digital signal in a temporary external memory storage device.
5. Send the digital signal from external memory into the DSK for processing.
6. Send the processed digital signal from the DSK into another temporary external memory storage device.
7. Convert the digital signal back to an analog signal.
8. Filter and apply a gain to the analog signal.
9. Transmit the manipulated signal.

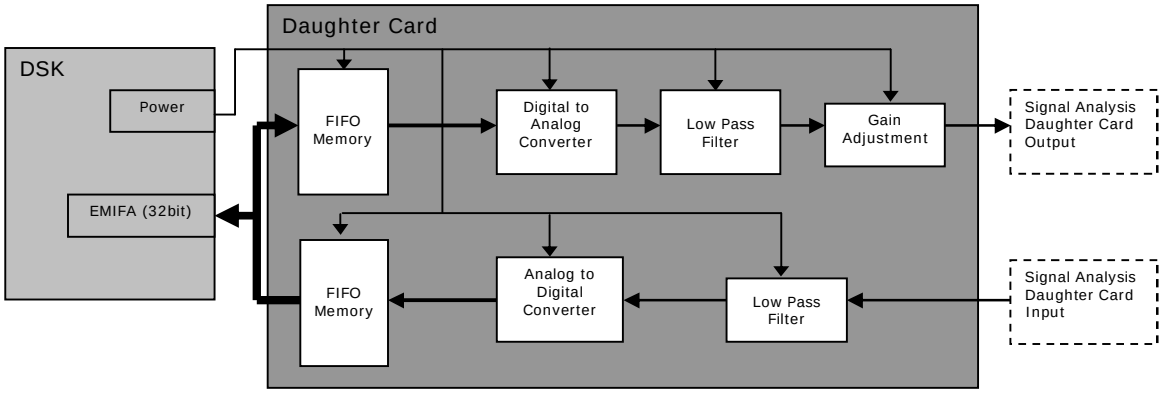


Figure 3.1 Block Diagram of Daughter Card

3.2 Required Features

- Convert two channels of analog signal into digital data.
- Buffer these channels in order for the DSK to access them.
- Allow for this data to be transmitted to the DSK's SDRAM
- Receive data from the DSK and transmit it out as an analog signal.
- The data resolution should be 12 bits at a conversion speed up to the max of the DSK.
- Operate near the maximum transfer rate of the EMIF.

3.3 Desired Features

- Easy to set up and use
    - The daughter card must be easy to connect to the DSK.
    - The software should be set up in such a way that it is easy to understand and edit.
- Variable conversion rate
    - This will allow the researcher to capture the experiment with different signals.
- Relatively Inexpensive
    - Compared to alternatives, this board should provide all features necessary, but for a cost that does not prohibit purchase or use.

Chapter 4 – Proposed Design and Options Considered

4.1 Introduction

Creating a board of this complexity requires many serious decisions that need to be executed in a timely manner. Many different components need to be selected. These include the high frequency data converters, a temporary memory storage device, and many other minor components that can play a big role in the success of this project. After selecting the proper components, they need to be created in the design software that we choose.

4.2 Analog Signal Acquisition

The board will need to be able to capture two analog signals as input. These signals will be of varying frequency, so it will be necessary to perform the data conversion as fast as possible. This data conversion requires an *Analog to Digital Converter* (ADC).

Some requirements of the ADC:

- Self contained, simultaneous sample and hold
- Sampling rates up to 100M samples per second
- Minimum of 12 bits of resolution
- If possible, more than one channel per chip

There are many different ADCs available. Because of the documentation available and ease of obtaining samples, it was decided that Texas Instruments (TI) should be chosen as the company. While sorting through the available products, it was noted that only one part fit our requirements. The ADS5521 ADC is a 125M samples per second, 12 bit, single channel converter. Not only does it meet the above requirements, but it also has an evaluation module that will be very helpful in designing the schematic and PCB.

4.3 Analog Signal Transmission

The daughter card will again need to be able to transmit two analog signals as output. The data from the DSK will have to be converted from a *Digital to Analog Converter* (DAC) at a very high frequency rate.

Some requirements of the DAC:

- Simultaneous output
- Conversion rate of up to 100M samples per second
- Minimum of 12 bits of resolution
- If possible, more than one channel per chip

The DAC that was chosen was the DAC2902. This DAC was chosen for multiple reasons. It is another TI part that contains useful documentation. The DAC2902 is a dual channel converter with a conversion rate of 105M samples per second. This will eliminate timing issues with the two channels running simultaneously. This part also has an evaluation module that will be helpful in designing the schematic and PCB.

4.4 Buffering (Input)

Buffering will be needed in this application so the DSP is not overloaded with information. There were many data storage components considered:

- The first storage device considered was *Electrically Erasable Programmable Read Only Memory* or EEPROM. It is a type of non-volatile storage that features byte-level writing. Each byte must be written or erased individually. This makes EEPROM very slow, where this design requires speed optimization. It also requires a high voltage input to program thus made it a very unsuitable choice.

- The second choice for data storage was *Synchronous Dynamic Random Access Memory* or SDRAM. This type of buffer requires a memory manager that would not be suitable for temporary storage. This would also require extra hardware to implement. The SDRAM also does not have a "burst mode" which is desired to receive and send the data.

- The third option considered was *First In First Out* memory or FIFOs. There are two main types of FIFOs. The first is synchronous and the second is asynchronous. The difference between the two types is the way in which they are clocked. Synchronous FIFOs use one clock rate for both the input and the output, whereas, asynchronous FIFOs can have the input clocked at a different rate from the output. The second option was more desirable for this project since the ADC will be able to clock the data in at a very fast rate, but the microprocessor will not be able to maintain the high rate of transfer for the FIFOs.

The part chosen was Texas Instrument's SN74V293 FIFO. It is a 65536 x 18 bit wide, 166MHz maximum frequency, asynchronous FIFO. It was also chosen for its broad range of technical documents including documents instructing connection to a DSK. This particular part was also easily available.

4.5 Low Pass Filters

The filter chosen for the input and output of the daughter card is a fourth order, 2-stage Chebychev filter with .5dB of ripple. The architecture is Sallen-Key. The cutoff is 30MHz. TI Filter Pro was used to calculate the values for the components. For the filter, E12 series resistors and E6 series capacitors were used. This is because it is easier to find through-hole components in house for a series with few components. The R1 seed in the program was set to 1k. This value sets the relative values of the components. A smaller R1 increases capacitor values, while a larger value increases resistor values. Setting R1 to 1k provides a nice balance where all components are in a reasonable range.

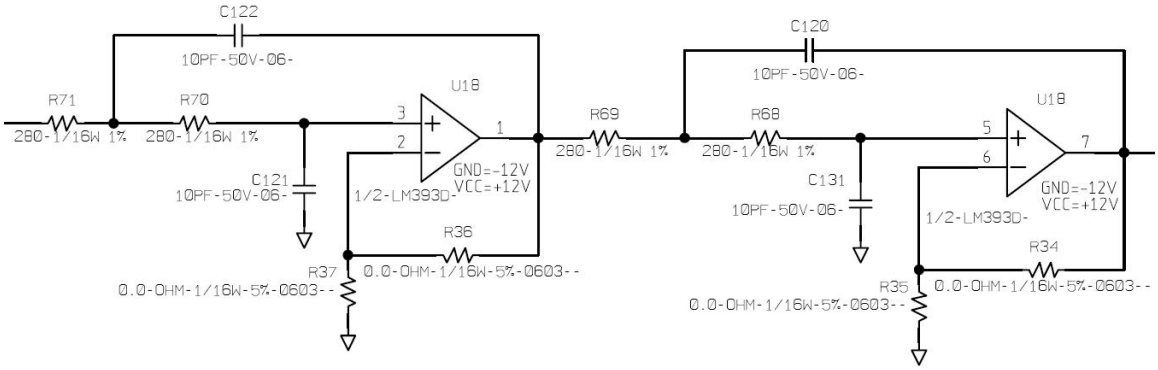
The transfer function for an individual stage looks like this:

$$\frac{V_o}{V_i} = \frac{K}{s^2(R_1R_2C_1C_2) + s(R_1C_1 + R_2C_1 + R_1C_2(1-K)) + 1}$$

where,

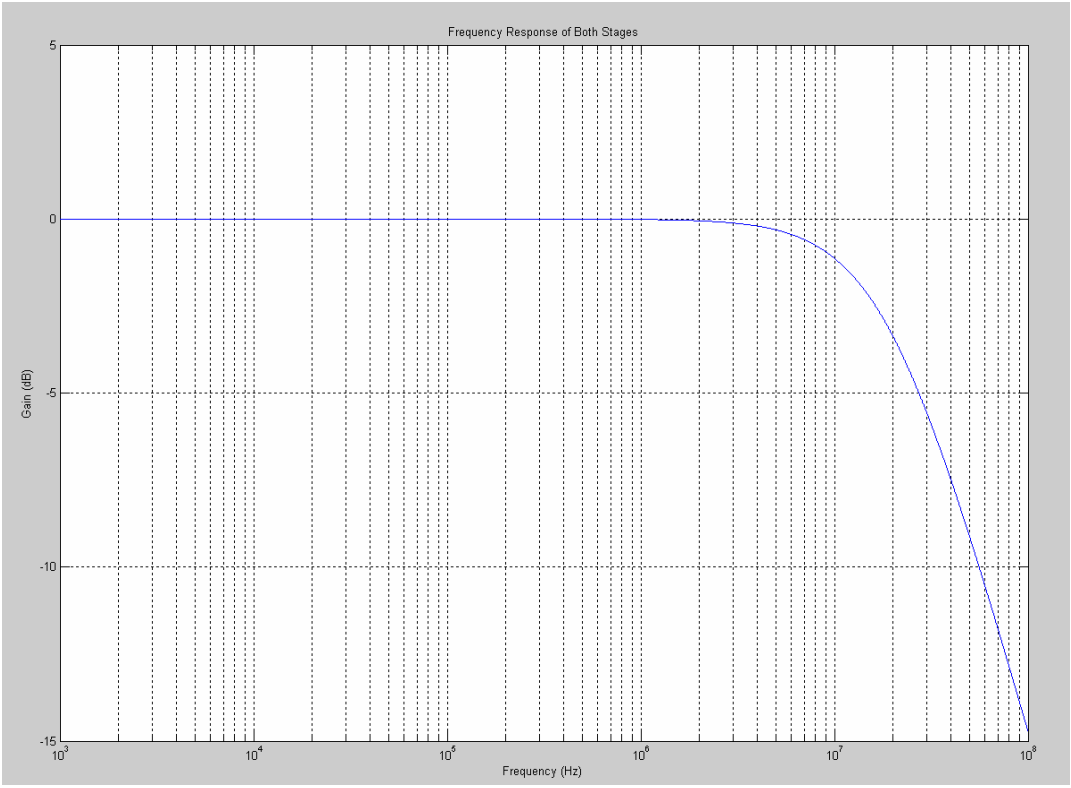
$$K = 1 + \frac{R_4}{R_3}$$

This is what the circuit looks like in the schematic:



Note: R36 and R34 are shorted, and R37 and R35 are left open, creating a gain of one.

Plot of the frequency response of the whole filter:



Chapter 5 – Hardware Design

5.1 Using the Daughter Card Connectors

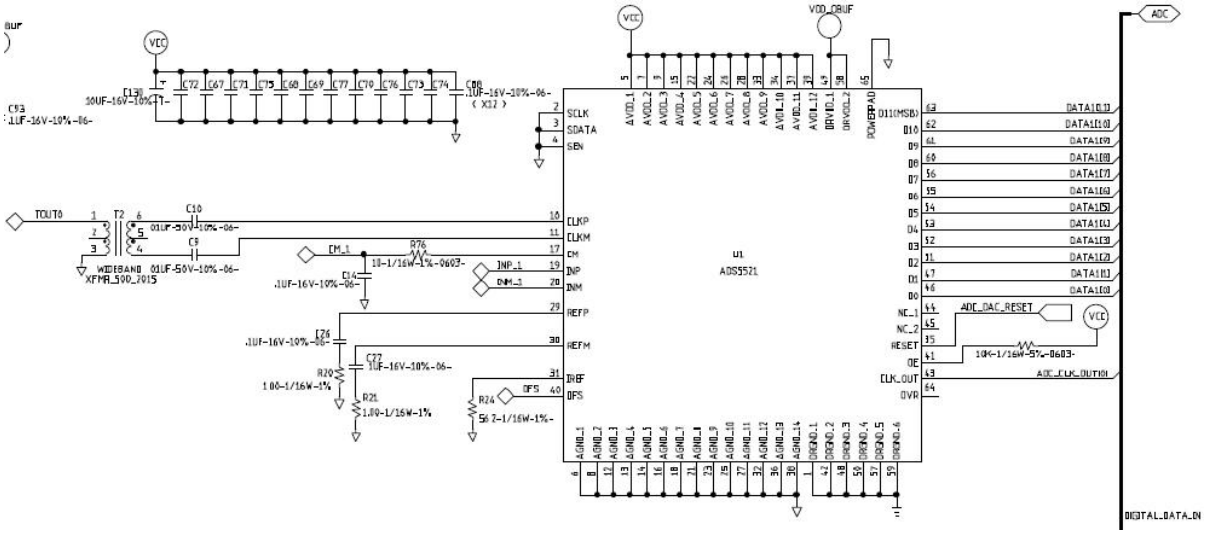
The daughter card connects to the DSK via the EMIF (J4) and the Peripheral Expansion Connector (J3). Through these two connectors, the daughter card receives power, clock signals, control signals, send and receive data, and trigger interrupts.

The major steps necessary to get the daughter card to work with these connectors is to create them in Mentor Graphics and position them correctly so that the board will fit. These measurements were taken from the data sheets for the DSK, which included a section on creating a daughter card.

As seen in Appendix B, the parts that were created provide a way of associating the connectors to other facets of the board.

5.2 Using the ADC

While designing the schematic for the ADC, many of the connections were made similar to the *Evaluation Module* (EVM). The pins were first researched in the data sheet in order to find information about their function. Then, the EVM was consulted. This usually provided further insight into the pin's function. Lastly, the pin was connected in the way that was best suited for this project. By designing in this fashion, many pins that controlled functions that were not in use could either be tied to either power or ground, reducing the need to use up valuable control pins from the DSK.



The above figure demonstrates some of the complexity that was involved in designing the board. It can also be noted that many pins are tied either high or low. The `TOUT0` clock line is tied to the clock input of the ADC using a method prescribed by the data sheet. The ADC has a clock out that is in phase with the data stream. There are separate circuits that filter and convert the input from single ended to differential mode. All of the power pins are filtered through a bypass capacitor. The output data is then sent to the FIFO memory storage.

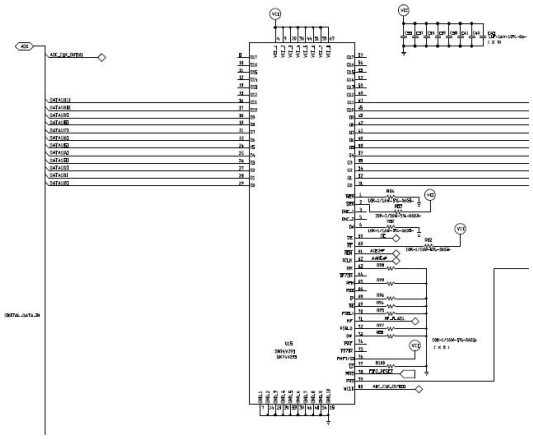
5.3 Interfacing the ADC, DAC, & FIFOs

In order for data to be transmitted to and from the external memory, several steps had to be accomplished. First, the data sheets were referenced. This provided the majority or the technical requirements for connecting these components. After this, the remaining steps included deciding on data modes and connecting the parts to facilitate these modes.

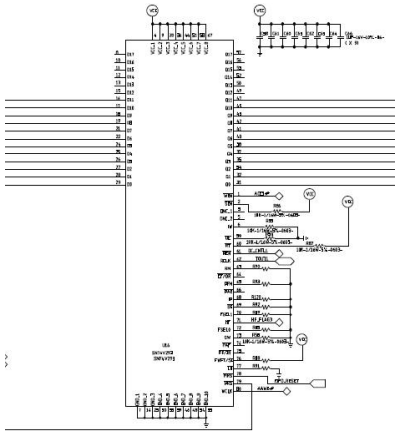
The FIFOs are set up in an asynchronous mode using First Word Fall Through (FWFT). With this DSK, the FIFOs cannot use synchronous mode. Synchronous mode is where the input and output clocks are driven from the same line. Asynchronous mode is where the clocks are driven from different lines. For this project, the input and output clocks are the same frequency, but they are at different phase angles, creating the need for asynchronous mode.

Due to the ADC having a clock output that is in phase with the data stream, it is used to clock the data into the input FIFOs. The `AARE#` pin from the DSK contains the clock signal that it uses to transmit data into internal memory. The `AAWE#` pin from the DSK contains the clock signal that is used to write the data from the DSK to the FIFOs and `TOUT1` is used as the read clock to output the data to the DAC.

The other control lines used were `ACE2#`, `ACE3#`, `AAOE#`, and `DC_CNTL0`. The one interrupt that the input FIFOs generate occurs when they are half-full. When either FIFO generates a half-full flag, INPUT_HF_FLAG is sent high.



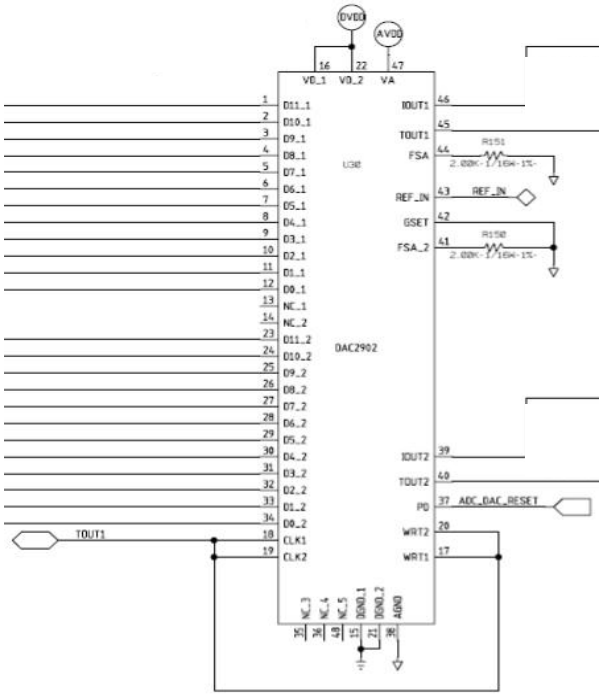
Input FIFO



Output FIFO

5.4 Using the DAC

The DAC schematic is set up in a manner similar to the ADC. The EVM was referenced and compared to the data sheet. The best configuration for this project was then selected. The clock inputs of the two channels are tied together so that they would be in phase. There is also an external voltage reference circuit so that the output could be calibrated to what is desired by the user. The circuits following the DAC convert the signal back to a single ended output, which is then filtered and transmitted.



5.5 Interfacing the FIFOs and the DSK

Connector	Pin	Description
J4	11,12,31,32,51,52,61,62,71,72,79,80	System ground
J4	41,42	3.3V voltage supply
J4	37-40,43-50,57-60,63-70	EMIF data pins
J4	73	EMIF async read enable
J4	74	EMIF async write enable
J3	1	+12V voltage supply
J3	2	-12V voltage supply
J3	3,4,7,8,25,26,31,32,37,38,43,44,51,52,61,62	System ground
J3	45	TOUT0
J3	49	TOUT1
J3	48	External interrupt 5
J3	53	External interrupt 4
J3	64	Daughter card crtl 0

5.6 Schematic Layout

The program chosen for creation of the schematic was DesignView by Mentor Graphics. This software is very user friendly and allows for smooth integration from a schematic to PCB. It also contains many other tools that aid in the development of the project.

Other Features of Mentor Graphics

- A CDB to BOM tool. This feature takes the current schematic and creates a Bill of Materials or BOM that states all of the parts used. This makes ordering parts and quantities very simple.
- Creation of tech maps. There is an ability to create a "top level assembly" and a "bottom level assembly". These show the placement of components on the PCB.
- A high-level viewer of the PCB that allows the viewer to highlight signals and view paths that signals take to various components.

5.7 PCB Layout

The program chosen for creation of the PCB was Expedition. This software goes hand in hand with DesignView. The user is able to highlight circuits within DesignView and each component within Expedition is then highlighted. This allows for grouping appropriate circuits and the corresponding components on the board.

Features of the PCB

- The board has to fit onto the DSK. This means that it is of an appropriate size to conform to the looks of the DSK. It also is able to fit in each mounting hole already used to secure the DSK. Size is also a factor when dealing with prices of PCB fabrication. Prices are set according to board area. The board is designed to be under 20 sq. in. to place it in a cheaper area category.
- The board is designed so that there is a spatial and chronological order to the process. The board follows the steps of the process of time reversal in an orderly fashion.
- There is a need for separate grounds on the board. Some components need to be connected to a ground that only has digital components grounded to it. Other parts need to be connected to a ground that only has analog components connected to it. This allows for less noise among components. The difficulty with the design of this is that the two grounds are moated so that they are isolated from each other, but are bridged together at a location on the board that is not affected by noise. The other problem with this is that because there is an isolation of grounds, the components are placed in an order so that they can be connected to an appropriate ground.
- The traces between components are run so that there is harmony between data signals. If one trace is run longer than another for a data line, one data line may have a longer time delay than another.
- Instead of test points, this PCB has open vias that accommodate probing for most important signals. Open vias have a small diameter of copper exposed around the fillet (the via hole) that can be easily probed.
- The board is a four-layer board. The board has two signal layers (top and bottom), a ground plane (AGND and GND), and a VCC plane.

Chapter 6 – Software Implementation

6.1 Background Information

The daughter card designed is meant to interface with a DSK. In order to do this interfacing, we must implement code that can be compiled and run on the DSK. The operation of the code works by taking input from the FIFOs, labeled corresponding to a read from the EMIF, and sending data to the output FIFOs, labeled corresponding to sending data to the EMIF.

The code that is used in this project is contained in three project files that are labeled read_test, write_test, and timereversal. The read test is designed to isolate the input side of the daughter card so that the user can strictly test the data coming in, without worrying about the operation of the output side. Similarly, the write test is designed to isolate the output side for testing. Write test merely generates a signal that acts as the input to be sent to the output side of the daughter card. Timereversal then, would be an attempt at combining the input with the output to be able to send a signal in and see the corresponding output, in the same program.

The following includes a descriptive overview of what the code does including sample code and flow charts that detail the operation of it. Also, the various parts of the daughter card that the code is interfacing to will be explained, including any control lines that are used. All of this will begin with functions of the DSK that control with the EMIF using *External Direct Memory Access* (EDMA).

It should be noted here that the code used for this senior design project is an extension of a previous senior design group that worked on the same project. The initial code development used was primarily written by Jonathon Kotta. It made little sense to rewrite the code from scratch since the designs of the two daughter cards were so closely related in high-level design and Jonathon had already handled the proper structure of the code to allow for configuration of the EDMA and the EMIF. Also, it had a proper mechanism for switching between modes of operation for the daughter card. Therefore, we had a head start on our code in terms of a structure to work with. This code, however, was modified to meet the needs of the current project, including setting up any controls lines that we had used and to properly configure the EMIF, which will be explained later, such that we can get the best transfer rate to and from the daughter card. Also, as of the time of the writing of this project, the main file, *timereversal.c*, which is supposed to allow for both input and output, has not been changed to reflect our project since we are still trying to get the input working using the test project read_test. We have seen proper output using the *write_test* function, *and timereversal.c* is merely a collaboration of these two functions. This can easily be seen by examining the attached code associated with our project.

The following tries to highlight the properties of the software related to this project. It highlights the main program, which as mentioned above, was primarily done by Jonathon Kotta and not extensively used for this project due to the debugging of the input of the daughter card. However, the highlighting of the main project explicitly details the

operation of both the test functions. *Remark:* The flow charts presented in the following pages referring to particular functions of the main program *timereversal* are taken from the previous senior design group, group 222, since the operation of the functions themselves have not changed.

6.2 Code Composer Studio

Code that is written and compiled for use on the DSK TMS320C6416 is done in a software package called Code Composer Studio. Projects that are built containing the code are organized for the software in a .pjt file, the project file. All of the code is contained in a single folder and there are two important files to consider. The first is the .c file that is the actual code being compiled to run on the DSK. Also, there is a .cdb file that is a configuration file for data setup by the DSP BIOS configuration utility included Code Composer Studio.

The configuration file is a configuration database used by CCS that is designed to simplify the task of configuring the setup of the DSK. However, for the majority of functions used in the operation of this code, this form of configuration was not used because it was easier to initialize dynamically within the code. Also, by performing the setup in the code, it was easier to see what was actually being done than what was being 'automatically' done by this configuration. It is used, however, in configuring interrupts and for establishing logs, such as a trace log used in the code. The interrupt that had to be configured, that saved a lot of effort from trying to configure it in code, is a `main_loop_tsk`, defined under the task section of the DSP/BIOS configuration, used to handle the interrupt that occurs once the `main()` function has finished and exited within the program. Logging was also easily configured by the use of it's logging features, where information, used for reason such as debugging or tracking the current state of operation within the DSK, can be done with the LOG_printf() function. This is equivalent to the standard c function printf(), used in conjunction with CCS.

EDMA is a great example of a function not configured by the configuration database. This was however performed with the use of the *chip support library* (CSL), which has a good API defined for use with the 6416. The API is clearly defined in the TMS320C6000 Chip Support Library API Reference Guide (SPRU401J.pdf). This document also includes other libraries that can be used as well with an overview of all register in the DSK and functions to manipulate these registers.

Components of the CSL are referred to as a module. All CSL modules are contained in a header file that defines symbol constants used with in the code. This allows for easy abstraction of the code that could easily be manipulated when certain portion's change, and allows for easier readability of the code itself. This header files are listed as a csl_*module*hal.h file, where *module* refers to the module that is being used. Since this library is being used in the code, it will be contained with in the listing of the header files contained in the project tree of CCS, generally found on the left hand side of the screen.

Once the code has been written and loaded on to the DSK, CCS provides utilities that can allow for debugging the code or just analyzing the operation of the DSK. Some of the

more important features used in this project are source stepping, assembly stepping and the viewing of memory on the DSK. Source stepping and assembly stepping refer to ability to run the code line by line, whether it be a function in the source (source stepping), or running each assembly instruction (assembly stepping). The operation of stepping is done with several keys or by using a drop down menu labeled *Debug*. By doing this, you can step through the source (F10), step into functions (F8), step out of functions (F7), run the code, at least until a break point has been incurred, (F5), and run to cursor, which runs all the instructions, as the source would follow itself , or until the user reaches the position of the cursor (F5).

Memory can be viewed in either a block of memory defined by a reference address, or by a variable with a quick watch function of CCS. Quick watch provide a simple interface to view both local variables of a function or other variables that you have defined by simply typing in the name of the variable in the quick watch window which displays it's contents. In addition, you can add a variable to be viewed, including arrays, by simply highlighting a variable name, right clicking on it, and selecting quick watch.

Also, by going to view -> graph -> time/frequency, CCS will plot data form memory. This was used for testing purposes by being able to see the data in a buffer, relative to what it might look like on an oscilloscope. This function works by plotting points in a buffer over a given range, such that the value in the buffer corresponds to the amplitude, and the position of the buffer corresponds to a particular moment in time. The utility allows you to setup up a configured frequency for properly labeled axes, although the plot itself is generally of more importance.

6.3 Timers for the DAC and ADC Clock

Both the input and output side of the daughter card must be driven by a clock controlled by the DSK. The daughter card uses two independent clock lines, labeled `TMR0` and `TMR1`. Each timer is a 50% duty cycle square wave that acts as a clock. In our implementation, `TMR0` controls the input side of the daughter card, and therefore is tied to the ADC. `TMR1` is used in conjunction with the output side of the daughter card, and is tied to the DAC. The frequency of these timers is controlled by a defined constant in the code called `SAMP_FREQ`. Although it would be nice if these timer's were configured directly by the frequency desire, it should be noted that they are actually configured by the period duration of the clock frequency. The formula for relating the output clock frequency to the period used to setup up the timers is defined in the code that it is automatically handled by the definition of the `SAMP_FREQ`. The period though is quantized corresponding to possible period values, and therefore there are a discrete set of frequencies that can be used, which are relative to the clock frequency of the DSK itself. At maximal operation, `TMR_PRD`, the value used to configure the timers, can be set to a value of 0, which on the 6416 allows for operation at 90 MHz. An important note is that if the period manager of the configuration database is on, that there is a problem of it trying to override `TOUT0`, or an equivalent pin, if not configured properly.

6.4 Interfacing the EMIF with FIFOs

Data sent and received from the daughter card is controlled on the EMIF. On the DSP 6416, this is referenced as *External Memory Interface A* (EMIFA). Although the interface with the FIFOs is synchronous, the actual interface in code of the EMIF refers to an asynchronous configuration that is configured based on the number of clock cycles for values that simulate the appearance of an asynchronous transfer. There are three values of particular interest called the setup, strobe, and hold values. Documents such as [10] and [5] explain this interface and discuss how to determine the proper values for setup, strobe, and hold so that the EMIF can be properly configured with the external memory source. The parameters for the EMIF are configured with the `set_emif()` function. Further, the values used in this project, including an overview of all relevant timing parameters can be found in appendix.

When reading data from the EMIF, since 12-bit resolution is being provided by the ADC and DAC, of the 32 data pins, only 24 were used. Pins [11,0] correspond to channel 1 with pin 11 being the most significant bit, while pins [27,16] correspond to channel 2 with pin 27 being the most significant bit. This configuration was chosen to be the most optimal, assuming both optimization of low-level assembly instructions, and based on the ease of use in programming. Given a 32-bit integer sample, containing both channels read from the EMIF, the code could restrict itself to separating the 32-bit integer into two 16-bit samples, and mask off the unused bits to get a valid data sample for the corresponding channel. This method also eliminates the need for possibly costly shifting operations, depending on the demand of the CPU when more extensive processing functions are added because of future improvements.

6.5 EDMA Transfers

Enhanced Direct Memory Access (EDMA) allows for a defined data transfer that is interrupt driven, limiting any wasted clock cycles that would be needed to otherwise check for data placed on the EMIF. For our configuration purposes, the FIFOs data is read like an address in memory; only the location to the address has been mapped in the memory. The source code defines two EDMA channels, used for input and output. Each channel is configured to trigger on an event. This defined event is what synchronizes the EDMA to the transfer through the EMIF. Since it is known that the flow of data that is coming into the EMIF is going to be a block of data at regular intervals, it makes sense to configure the use of EDMA, which handles all of the data transfer into a buffer, or other memory location, on its own.

When an event is triggered for the EDMA, the EDMA channel signals a transfer request, at which point the channel is given data by an EDMA controller. When the EDMA completes a data transfer it signals an `EDMA_Controller` interrupt. Ideally, the main program *timereversal.c* should have EDMA events triggered by interrupts 4 and 5, which are signaled by the half full flag of the input and output FIFOs. These events, however, are net yet functional for the current daughter card, since there is (at the time of the writing) still debugging at the input from the daughter card. There is the ability to tie the triggering event to a timer.

When an EDMA interrupt is triggered, the code runs an `edma_hwisr()` function. This function checks whether the stack used for transferring data is full, configures the DSK for the next transfer, or changes the mode of operation of the daughter card, depending on whether it needs to be in a read mode or write mode, which is explained later in this section. The EDMA configuration of this code is handled dynamically as mentioned previously through the use of two function `set_read_mode()` and `set_write_mode()` based on the daughter cards mode of operation, which use `EDMA_config()` to configure the EMDA with the proper settings. `EDMA_config()` is passed an `EDMA_config` structure that is defined in a header file, in accordance with the DSK memory structure. `EDMA_config` structures contain all of the parameters that are defined for an EDMA transfer. The primary difference between the `EDMA_config` structure passed when in read mode or write is the source and destination addresses of the transfer. For input, the source address will be the FIFO tied to the EMIF, with a destination address referencing a buffer in the memory of the DSK. For the output, it is the opposite, where source data is referenced from a buffer in the memory of the DSK out to the FIFOs. This source data will be the process data taken from the input.

Documentation on the EDMA can be found, and explained in detail, in [14], which is the complete reference guide to programming the DSK for EDMA with examples and explanations, and [16], which is more of a quick reference guide for the EDMA controller. The previous senior design group references [4], an extensively detailed reference guide on the EDMA architecture, used in the development of the code.

6.6 Modes of Operation

The code, and equivalently the daughter card, has two modes of operation, which in the code are considered read mode and write mode. Read mode refers to the reading of information input to the daughter cards and set through the EMIF to the DSK. Write mode refers to the sending of data through the EMIF to the output side of the daughter card. The state of the program is defined by a register that acts as a flag called `read_mode` in *timereversal.c*. The programs used for testing entitled `write_test` and `read_test` equivalently handle these two modes independently, without the need for switching between the two. Therefore, the operations performed while in read mode of the main program are performed in the `read_test` project, and operations performed while in write mode of the main program are performed in the `write_test` project. Thus, as mentioned above in the background, the main program combines these two functions.

When the flag `read_mode` is set to a value of true, in terms of a Boolean expression, the program is in a read mode, while if the value is false, we are in write mode. These modes are configured at runtime, although will by default be started in read mode based on definitions established in the code. This allows for the possibility of the code to be started in write mode, but due to the applications of this project, is not likely a desired feature, or of much use since no data will have as of yet been read in. Therefore, this is primarily a function of debugging. The constant defined for the starting mode of operation is `START_IN_WRITE_MODE`.

As mentioned above, the main programs used for this project were `read_test` and `write_test`, however similar operation to both of these can be found in the main program by removing the ability of the main program to switch between modes. Since the switching of modes is handled by the EDMA interrupt service routine `EDMA_hwisr()`, by setting the control statement for changing modes to 0, one can stick to either read mode or write mode based on the above `START_IN_WRITE_MODE` declaration. For the purposes of debugging the board, it was more advantageous to keep the two distinctly separated for the time being until the two parts of the board were working first.

6.7 Buffers

The configured EDMA transfers for data sent on the EMIF use what is commonly referred to as ping-pong buffers. These ping-pong buffers are a set of buffers, where the active buffer receives data until it is full, at which point the active buffer is switched to another ping-pong buffer, and the now full buffer can be processed if desired. The operation of the code contains two pointers, `pp_read_ptr` and `pp_write_ptr`, which refer to the current address for reading and writing to the buffers. The pointers are adjust by corresponding routines for reading and writing, i.e. only functions that write to the buffers should adjust the write pointer, and only function that read from the buffers should adjust the read pointer.

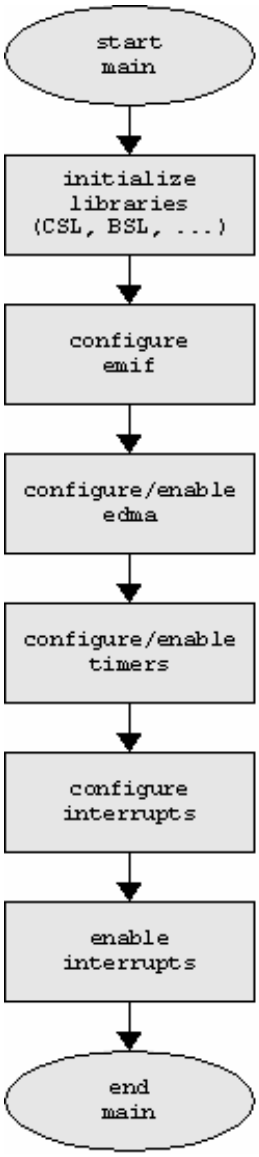
When in write mode, the buffers work by allowing the pointers to point to a particular frame, or index of the buffer being used. If at any time the read pointer and the write pointer are pointing to the same address, the assumption is that all the elements of the buffer are full. The code will then either give an error message to the log file, or wait for an available frame. The pointers are incremented cyclically, modulo the number of frames, which in the code is defined by the constant `NUM_PINGPONG`. Similarly, in read mode, if the pointers are equivalent, it can be assumed that the buffers are read, or in other words, all of the data that has been input has been read. Again, the point is incremented as it is in write mode. The functions corresponding to handling the buffers is the `read_loop()` function and `write_loop()` function, which they themselves are handled by the interrupt service routine `edma_hwisr()`.

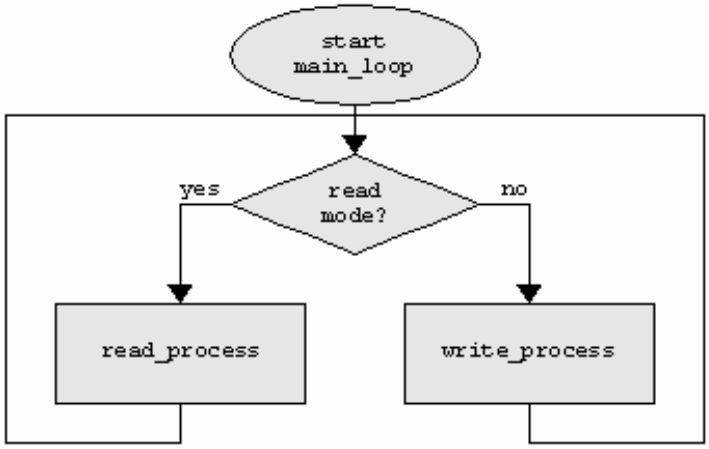
With the test projects, read test and write test, these buffers are not necessary and we can limit everything to having just a single buffer, (called `buffer[]`), which for the read test stores any data transferred from the input, and for the write test, stores a generated signal that is created during runtime.

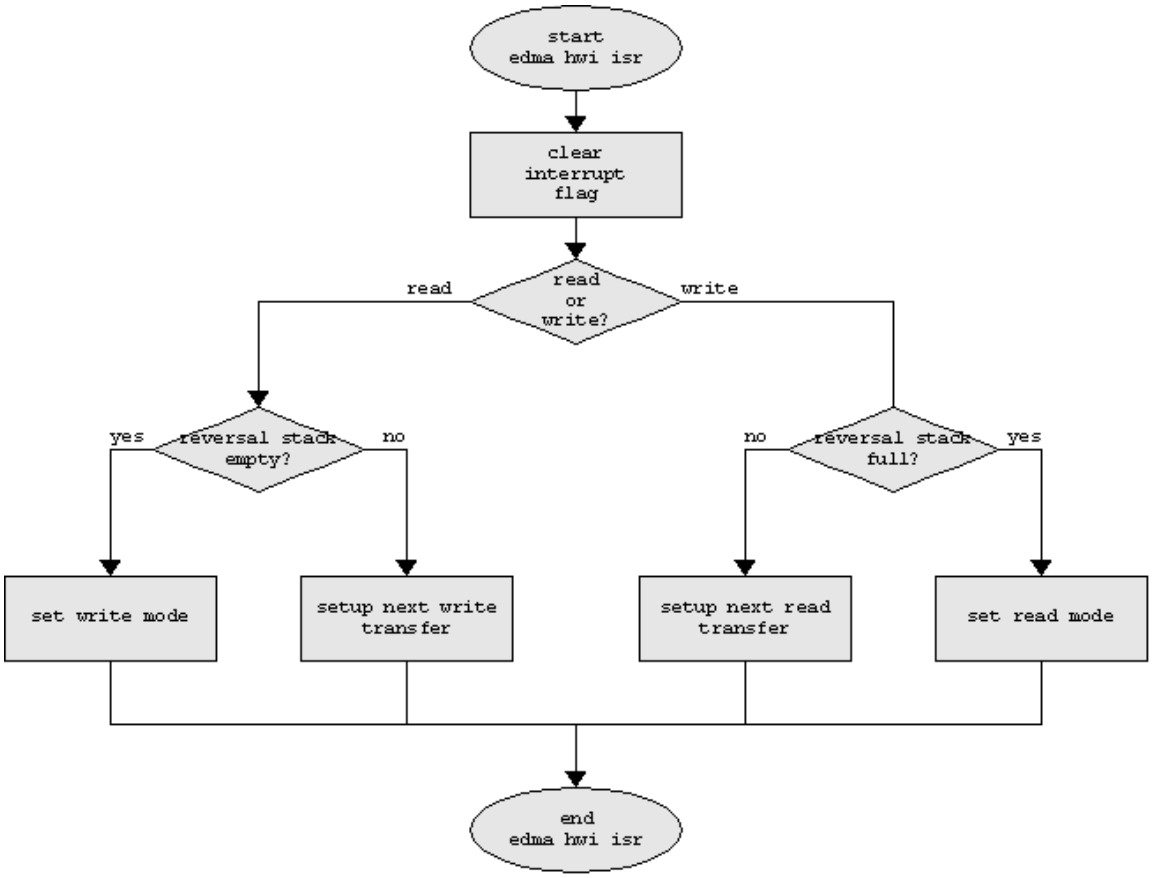
Another important buffer that is used is the reversal stack (`stack[]`). This is strictly limited to the operation of storing data samples, and is accessed by two small functions `push_rs()` and `pop_rs()`. These two functions server no other purpose other than to retrieve and store samples from this stack, both of which are common to any computer language and programming theory.

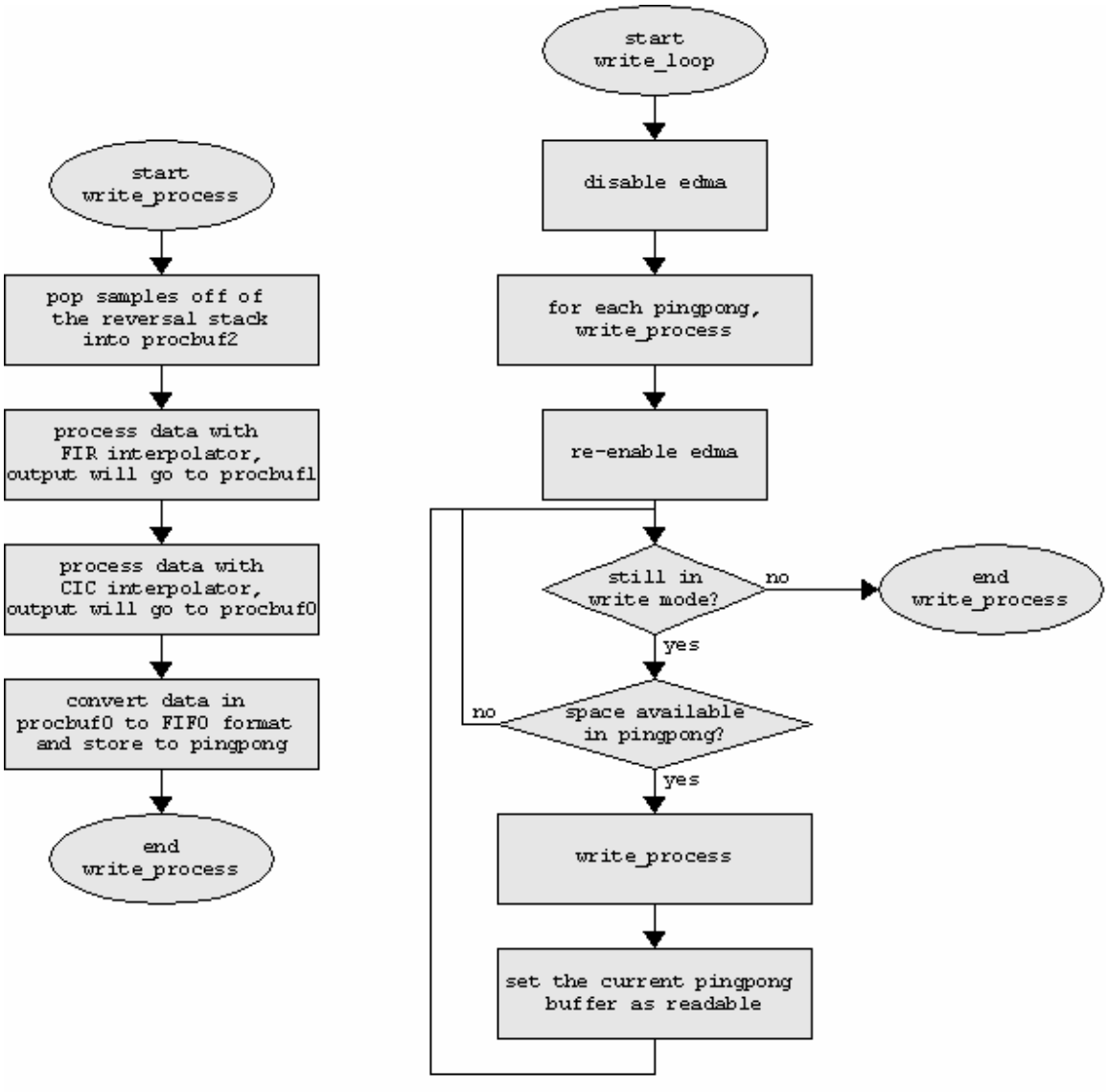
6.8 Flow Charts

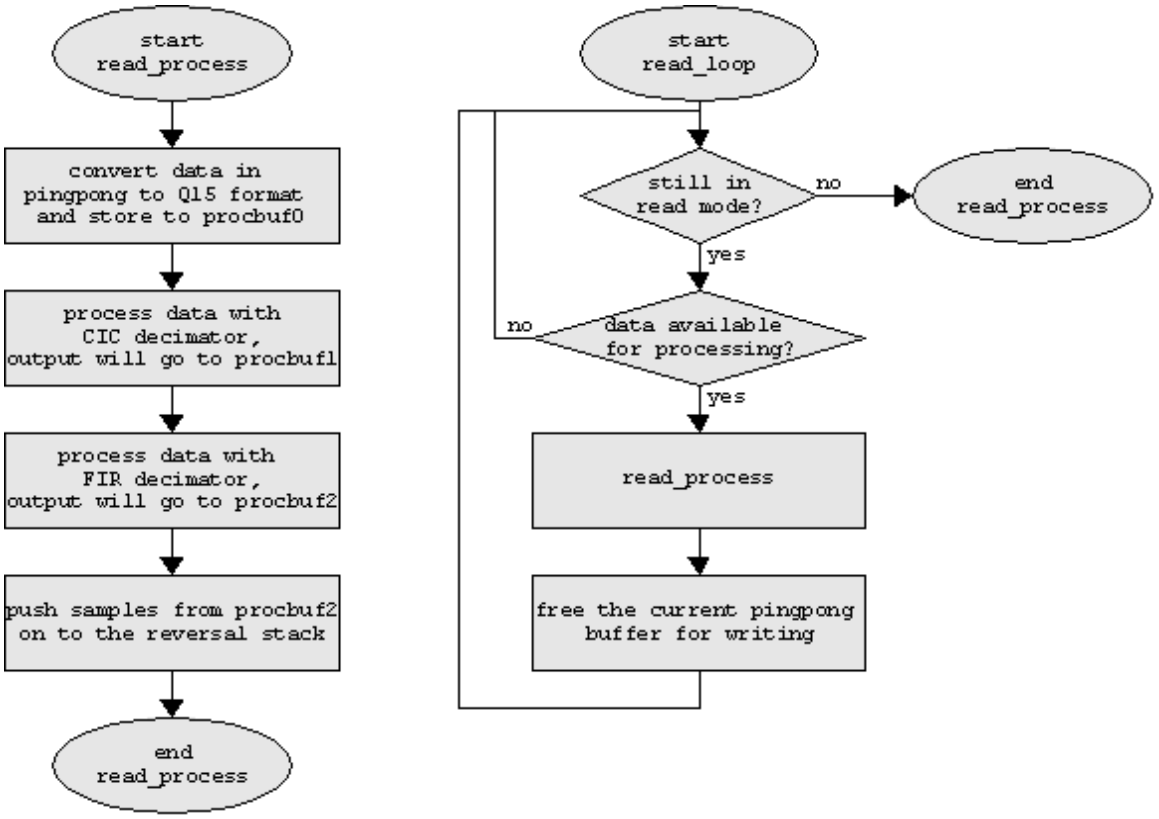
The following flow charts outline the operation of the functions in the main program *timereversal*. The `main()` function is primarily used for configuring the DSK and initializing variables. Once the `main()` function exits, the routine `main_loop()` runs, and controls the mode of operation of the DSK. The two test functions used, read test and write test, have the same configuration, but will not have the others function, i.e. read test does not contain `write_loop()`, and write test does not contain `read_loop()`. Also, the test functions do not contain different `EDMA_config` structures that need to be initialized, since the EDMA channels only need to be configured for one mode of operation.


start
main
initialize
libraries
(CSL, BSL, ...)
configure
emif
configure/enable
edma
configure/enable
timers
configure
interrupts
enable
interrupts
end
main


start
main_loop
yes
read
mode?
no
read_process
write_process


start
edma hwi isr
clear
interrupt
flag
read
or
write?
read
write
yes
reversal stack
empty?
no
no
reversal stack
full?
yes
set write mode
setup next write
transfer
setup next read
transfer
set read mode
end
edma hwi isr


start
write_process
pop samples off of
the reversal stack
into procbuf2
process data with
FIR interpolator,
output will go to procbuf1
process data with
CIC interpolator,
output will go to procbuf0
convert data in
procbuf0 to FIFO format
and store to pingpong
end
write_process
start
write_loop
disable edma
for each pingpong,
write_process
re-enable edma
still in
write mode?
no
end
write_process
yes
no
space available
in pingpong?
yes
write_process
set the current pingpong
buffer as readable



If one looks at the processing functions `read_process()` and `write_process()`, notice that there are other functions mentioned entitled decimator's and interpolator's. These processing functions have not been tested but would be appropriate for a fully functional time reversal system. They are however, not necessary for a proof of concept of verifying correct operation of input and output. Ideally, the simplest form of operation is to send any data read in directly to the output, without modification, with the idea that the exact same signal should come out.

Chapter 7 – Testing and Debugging

7.1 Introduction

After the daughter card was designed and sent out for fabrication, testing had to be done to ensure correct operation of the board. This first started out while the board was being populated. This included ensuring that each of the components that had to be soldered on to the board preserved continuity and that each pin of the individual components had the correct signals being sent to them. As will be shown, several problems were encountered during this stage of the design, and each required a different method to fix the problem. This will be outlined in this section, where the problems will be detailed with solutions for fixing it.

7.2 Problems Encountered

Remark: Each of the following problems encountered is listed in the order in which it was found during the debugging stage of this project.

1. Via connecting power to ground
2. Put EMIF connector on wrong
3. +5V J3 pins connected to ground
4. Capacitors on -12V supply were on backwards
5. Capacitors on +/-12V supply were not rated to handle 12V.
6. Period manager of DSP/BIOS configuration was overriding `TOUT0`, timer 0, leaving no clock.
7. ADC power pad connected to VCC. Should have been connected to analog ground and used as a heat sink.
8. PRS (partial reset switch), not set to a high value in the code, which is controlled by `DC_CNTL0`, part of the daughter card control register.
9. `TOUT0`, the clock for the input, was being loaded by the transformers when converting from single ended to differential on clock input to ADC. Removed transformer and used single ended.
10. Pins routed through the powerpad, whose traces were cut, had to be removed. This was not supplying VCC to some of the capacitors.
11. ADC overheated and shorted power to ground. The ADC was dead.

7.3 Debugging

All of the problems that were found, as listed in order above, were found through a methodology of testing. First, soldering on the components of the printed circuit board and performing continuity tests exposed several problems. The first problem, with a short from power to ground, was easily seen right away, after only a few of the components had been soldered on. Although this was worrisome at first, after reviewing the PCB layout and schematic design, it was determined that the only possible place that a short of this magnitude could have occurred, other than due to a manufacturing problem, was in our bridge that connects the analog ground to digital ground. It just so happened that this bridge was at the same location as a via which was Vcc. Therefore,

drilling out the via with an x-acto knife removed the short and the board could continue to be soldered.

The next couple of problems found were due to capacitors. This problem identified itself as smoke when the board was plugged in and powered up. This obviously signified that something was wrong and the location of the problem was clearly known. It was found that the polarized capacitors on the power filter for the -12V supply were soldered incorrectly. They were installed with the minus side pointing to ground, but for this supply, it should point to -12V. While diagnosing this problem, it was noticed that, as shown on the capacitors, they were not meant to handle the voltage that was being run across them. These were both two easy mistakes to fix that did not deter much from the project.

After these mistakes had been fixed, the board could then be placed onto the DSK and powered up without any problems. Therefore, testing of the functionality of the board began, in which the clocks that feed the ADC and DAC were tested. Here, it was noticed that one of the clocks was not outputting anything. This clock was TOUT0, which handles the components of the input side of the board. After reviewing the code, it was noticed that in the configuration database handled by the DSP BIOS, that the clock was being overridden by what is called the period manager. By properly configuring the DSP BIOS and deactivating the period manager, the clock problem was fixed, and it could now be verified that all the components were at least receiving a clock.

Next, testing of the input began. Here it was noticed that not only was the ADC running extremely hot, but it was barely functioning at all. Upon reviewing the data sheets, it was noticed that what is deemed as a "power pad," which is used for thermal dissipation, is actually supposed to be tied to ground and not to Vcc. This lead to a big work around, in which the power pad had to be isolated but cutting the traces connected to what was Vcc, and the rerouting the other traces. Also, this required a bridge from the power to analog ground, which was done by removing the soldermask and getting to the copper, which is analog ground on the top layer of the board where there are not any traces, and then putting a bridge from the power pad to the copper.

After the power pad was fixed, the output of the ADC was still not there. Here, it was noticed at the same time that the output enable pin, which was floating, was supposed to be tied to Vcc. However, it was thought at the time of designing the board that it had an internal pull up resistor and this was not necessary. This was not the case, but was fixed by tying the pin to the Vcc side of a bypass capacitor.

Next, the FIFOs, which are the external memory storage from the perspective of the DSK, were constantly being reset. This could only be due to the partial reset switch, which allows the memory contents to be cleared in the FIFO. The decision had been made to control this in the software, but it had not been accounted for in the software at this time. This only required setting a register the daughter card control register that set DC_CNTL0, bit 0, of the register to a value of 1 during the initialization of the DSK, when the software is loaded.

Also, the clock signals of the board, which had been fixed previously, were not looking good in the sense that they did not meet the expectations of what the clock signal should have looked like. The voltage on the clock signals was extremely low. This was due to the attenuation of the signal due to improper configuration of the transformers attached to the board that were designed to convert the clocks from single ended mode to differential. This also lead to determining that all of the transformers on the board were improperly configured and were attenuating most of the signals, at least on the output side of the daughter card.

Finally, as mentioned previously, the ADC had been running extremely hot and it was noticed that its functionality was not meeting expectations. Upon trying to fix other problems, the ADC finally died and created another short from power to ground. Although it was initially thought that this might have had something to do with our bypass capacitors due to the location of the short, which was thought to be isolated, it was later determined that the ADC itself had been fried and created the short to ground. Not so simply, by removing the ADC and replacing it with a new one the short was fixed.

All of these problems were found by systematically testing the functionality of the board. By stepping through the operation of the board, problems were able to be isolated. This is extremely important in a board of this complexity. Also, it teaches a lot about proper debugging skills, which are essential to any design engineer.

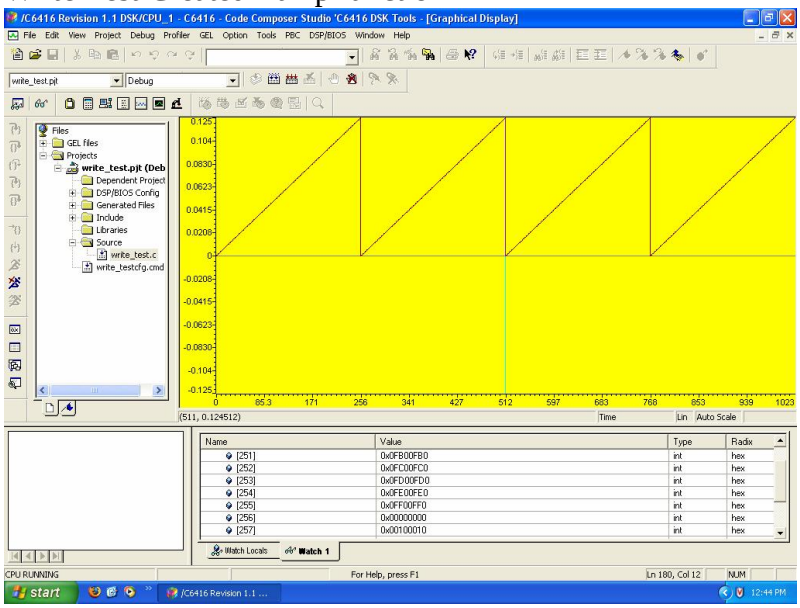
Although there still exists a problem that has of yet not been determined on the input side, regarding the communication from the ADC to the EMIF, which is most likely seen to be a cause of the ADC, this problem is again isolated. A method of testing is occurring at the time of this writing. This method is sure to determine the cause of the problem and allow for finalization of the project.

7.4 Tests

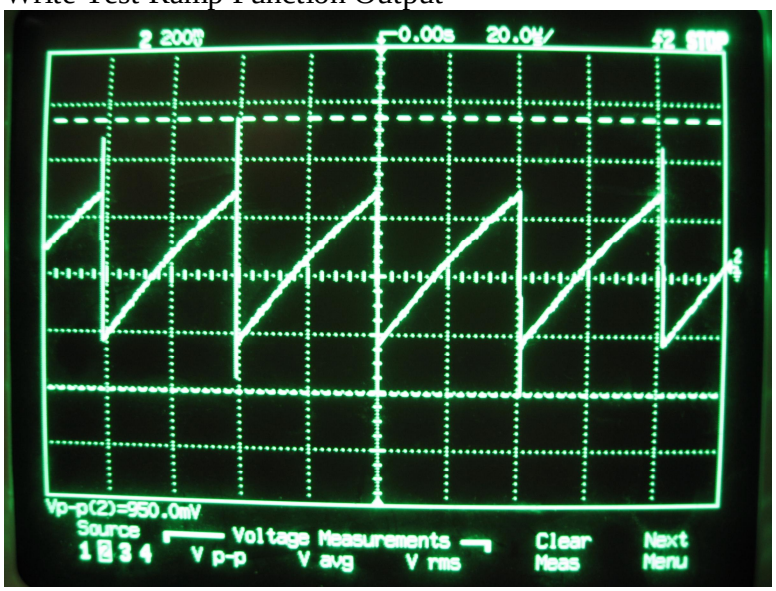
There are many easy places to test on the board to check for general functionality. Please see the Users Guide for easy access to important signals that will tell if the board is generally working properly.

- Read_Test
    - The read test is designed to isolate the input side of the daughter card so that the user can strictly test the data coming in, without worrying about the operation of the output side.
- Write_Test
    - The write test is designed to isolate the output side for testing. Write test merely generates a signal that acts as the input to be sent to the output side of the daughter card.

Write Test Created Ramp function



Write Test Ramp Function Output



Chapter 8 – Conclusion

8.1 Summary

This board may be used to analyze many aspects of high frequency signals. The intended purpose is to time reverse signals. It is a very good tool to use for acquiring and transmitting signals.

- There are many improvements that could be made to this design. Most of the problems that were encountered resulted in manual changes to the PCB. These problems could be avoided in the next rev of this board resulting in a highly simplified debugging process since most of the hardware errors will be eliminated.
- One way that this project could have been simplified would have been to set the requirements early in the project and understand them so that more time could have been devoted to debugging and testing.
- It is important to maintain documentation as this will be a tool later in the project.
- The college faculty is willing and able to aid in any problem encountered. It is highly recommended to utilize this knowledge base.

8.2 Budget

Description	Digikey Part Number	Quantity Needed	Total Price
Resistors			
0 Ohm 1/10W 5% 0603	RHM0.0GCT-ND	16	$1.86
1 Ohm 1/10W 1% 0603	311-1.00HRCT-ND	4	$1.54
10 Ohm 1/10W 1% 0603	RHM10.0HCT-ND	2	$1.52
24.9 Ohm 1/10W 1% 0603	RHM24.9HCT-ND	6	$1.52
36.5 Ohm 1/10W 1% 0603	RHM36.5HCT-ND	4	$1.52
49.9 Ohm 1/10W 1% 0603	RHM49.9HCT-ND	6	$1.52
56.2 Ohm 1/10W 1% 0603	RHM56.2HCT-ND	2	$1.52
100 Ohm 1/10W 1% 0603	RHM100HCT-ND	2	$1.52
174 Ohm 1/10W 1% 0603	RHM174HCT-ND	2	$1.52
1K Ohm 1/10W 1% 0603	RHM1.00KHCT-ND	5	$1.52
2K Ohm 1/10W 1% 0603	RHM2.00KHCT-ND	2	$1.52
10K Ohm 1/10W 1% 0603	RHM10.0KHCT-ND	49	$4.80
10K Ohm 1/10W 1% 0603 Extra	RHM10.0KHCT-ND	49	$1.52
500 Ohm Potentiometer 1/4W 20%	490-2654-1-ND	3	$6.74
Capacitors			
10pF 50V 5%	PCC100CVCT-ND	2	$0.62
27pF 100V 5%	490-1337-1-ND	2	$0.55
.01uF 16V 10%	PCC1750CT-ND	8	$0.70
.1uF 16V 10%	PCC1762CT-ND	85	$4.25
.1uF 16V 10% Extra	PCC1762CT-ND	85	$1.05
10uF 6.3V 20%	PCC2395CT-ND	1	$4.10
10uF 50V 20% Polarized	495-2259-1-ND	10	$6.63
47uF 10V 10% Polarized	495-2216-1-ND	6	$3.57
Filter Parts			
Resisters			
180 Ohm 1/10W 5% 0603	RHM180HCT-ND	4	$1.52
220 Ohm 1/10W 5% 0603	RHM220HCT-ND	4	$1.52
1.50k Ohm 1/10W 5% 0603	RHM1.50KHCT-ND	8	$2.28
Capacitors			
1pF 50V +/-.25pF	511-1088-1-ND	4	$1.02
6.8pF 50V +/-.25pF	490-1395-1-ND	4	$1.60
33pF 50V 5%	PCC330ACVCT-ND	4	$0.62
100pF 50V 5%	PCC101ACVCT-ND	4	$0.62
ICS			
TI SN74V293		4	FREE
ADS5521 12-Bit, 105 MHz ADC		2	FREE
2902 Dual Channel 12 Bit 125 MHz max. DAC	DAC2902Y/250CT-ND	1	$15.26

Ferrite Beads			
Ferrite Chip Signal 2000 Ohm SMD	240-2413-1-ND	7	$2.26
Transformers			
Coilcraft WB3010-SML .005-100 MHz 1:1		8	FREE
Voltage Reference			
TI 1004D-1.2		1	FREE
Opamps			
AD826 50MHz Dual		5	FREE
Gates			
OR Gate Single	296-1115-1-ND	1	$1.95
AND Gate Single	296-11601-1-ND	2	$1.95
PCB		3	$338.00
	Mouser Part Number		
Switches			
Dual SPST NO Switch SMD	512-FSA266K8X	1	$1.60
SPST NO Push Button	612-320.01E1-1WHT	1	$6.03
Connectors			
SAMTEC TFM-140-S-D-LC 80 Pin Connector		2	FREE
Molex 10-89-7161 32 Pin Dual Row Breakaway Connector		10	$8.80
SMA	523-901-143-6RFX	5	$53.40

Total Estimated Cost	$420.20

Chapter 9 - References

[1] ADS5500/5541/5542/5520/5521/5522 14- and 12-Bit Single Channel ADC EVM ADC Evaluation Module User's Manual. 2005. Available on the web at <http://focus.ti.com/lit/ug/slwu010c/slwu010c.pdf>.

[2] ADS5521 Datasheet. April 2006. Available on the web at <http://www-s.ti.com/sc/ds/ads5521.pdf>.

[3] Ardizzoni, John. A Practical Guide to High-Speed Printed-Circuit-Board Layout. September 2005. Available on the web at <http://www.analog.com/library/analogDialogue/ archives/39-09/layout.pdf>.

[4] Bowen, Jamon, and Jeffrey Ward. TMS320C64x EDMA Architecture. March 2004. Available on the web at < http://focus.ti.com/lit/an/spra994/spra994.pdf>.

[5] Castille, Kyle. TMS320C6000 EMIF to External FIFO Interface. May 1999. Available on the web at <http://focus.ti.com/lit/an/spra543/spra543.pdf>.

[6] DAC2902 Datasheet. November 2003. Available on the web at <http://www-s.ti.com/sc/ds/dac2902.pdf>.

[7] DAC290x-EVM DAC Evaluation Module User's Manual. September 2005. Available on the web at <http://focus.ti.com/lit/ug/sbau071b/sbau071b.pdf>.

[8] Dell, Dave. TMS320 Cross-Platform Daughter Card Specification Revision 1.0. November 2000. Available on the web at <http://focus.ti.com/lit/an/spra711/spra711.pdf>.

[9] DSP Starter Kit (DSK) for the TMS320C6416 (720 MHz) Quick Start Installation Guide. Available on the web at <http://c6000.spectrumdigital.com /dsk6416/V2/docs/dsk6416_QuickStartGuide.pdf>.

[10] Finger, Robert. Using TI FIFOs to Interface High-Speed Data Converters With TI TMS320™ DSPs. June 2001. Available on the web at <http://www-s.ti.com/sc/psheets/sdma003/sdma003.pdf>.

[11] Jones, Kevin. TMS320C6416 Hardware Designer's Resource Guide. October 2005. Available on the web at <http://focus.ti.com/lit/an/spra943b/spra943b.pdf>.

[12] Kotta, Jonathan, Eric Rossland, and Ryan Schumacher. "A Technical Description of a DSP Signal Conversion Card." Unpublished notes, 29 Apr. 2005. 24 Apr. 2006 <http://snrdes2.ece.ndsu.nodak.edu/~grp222/files/whitepaper_final.pdf>.

[13] SN74V263, SN74V273, SN74V283, SN74V293 8192 × 18, 16384 × 18, 32768 × 18, 65536 × 18 3.3-V CMOS FIRST-IN, FIRST-OUT MEMORIES. February 2003. Available on the web at <http://www-s.ti.com/sc/ds/sn74v293.pdf>.

[14] TMS320C6000 Chip Support Library API Reference Guide August 2004. Available on the web at <http://focus.ti.com/lit/ug/spru401j/spru401j.pdf>.

[15] TMS320C6000 Chip Support Library API User's Guide. October 2001. Available on the web at < http://rcv.kaist.ac.kr/lecture/ee308_2003/notes/spru401c_csl.pdf>.

[16] TMS320C6000 DSP Enhanced Direct Memory Access (EDMA) Controller Reference Guide. July 2003. Available on the web at < http://focus.ti.com/lit/ug/spru234b/spru234b.pdf>.

[17] TMS320C6000 DSP External Memory Interface (EMIF) Reference Guide. April 2005. Available on the web at <http://www.tij.co.jp/jsc/docs/dsps/support/download/c6000/c6000pdf/spru266d.pdf>.

[18] TMS320C6000 DSP External Memory Interface (EMIF) Reference Guide. February 2006. Available on the web at <http://focus.ti.com/lit/ug/spru266e/spru266e.pdf>.

[19] TMS320C6000 DSP Peripheral Component Interconnect (PCI) Reference Guide. June 2004. Available on the web at <http://focus.ti.com/lit/ug/spru581b/spru581b.pdf>.

[20] TMS320C6414, TMS320C6415, TMS320C6416 Fixed-Point Digital Signal Processors. May 2005. Available on the web at <http://focus.ti.com/lit/ds/symlink/tms320c6416.pdf>.

[21] TMS320C6416 DSK Technical Reference. November 2003. Available on the web at <http://c6000.spectrumdigital.com/dsk6416/V3/docs/dsk6416_TechRef.pdf>.

[22] TMS320C6x C Source Debugger User's Guide. January 1998. Available on the web at <http://focus.ti.com/lit/ug/spru188d/spru188d.pdf>.

Appendix A – Software Configuration

Software Configuration, Timing, and Calculated Values

Information taken from SPRU401j
ce2 controls input
ce3 controls output

EMIF Global Control Register (GBLCTL) Field Values

Pin Information				Configuration
Bit	**Field**	**Symbol Val**	**Value**	**Description**
31−20	Reserved		0	Reserved. The reserved bit location is always read as 0. A value written to this field has no effect.
19−18	EK2RATE	FULLCLK	0	1× EMIF input clock (ECLKIN, CPU/4 clock, or CPU/6 clock) rate.
17	EK2HZ	CLK	0	ECLKOUT2 continues clocking during Hold (if EK2EN = 1).
16	EK2EN‡	ENABLE	1	ECLKOUT2 is enabled to clock.
15−14	Reserved		0	Reserved. The reserved bit location is always read as 0. A value written to this field has no effect.
13	BRMODE	MRSTATUS	1	BUSREQ indicates memory access or refresh pending or in progress.
12	Reserved		0	Reserved. The reserved bit location is always read as 0. A value written to this field has no effect.
11	BUSREQ	LOW	0	BUSREQ output is low. No access/refresh pending.
10	ARDY	LOW	0	ARDY input is low. External device is not ready.
9	HOLD	LOW	0	HOLD input is low. External device requesting EMIF.
8	HOLDA	LOW	0	HOLDA output is low. External device owns EMIF.
7	NOHOLD	DISABLE	0	No hold is disabled. Hold requests via the HOLD input are acknowledged via the HOLDA output at the earliest possible time.
6	EK1HZ‡	CLK	0	ECLKOUT1 continues clocking during Hold (if EK1EN = 1).
5	EK1EN‡	ENABLE	1	ECLKOUT1 is enabled to clock.
4	CLK4EN	ENABLE	1	CLKOUT4 is enabled to clock.
3	CLK6EN	ENABLE	1	CLKOUT6 is enabled to clock.
2	Reserved		1	Reserved. The reserved bit location is always read as 0. A value written to this field has no effect.

1−0	Reserved		0	Reserved. The reserved bit location is always read as 0. A value written to this field has no effect.

EMIF CE2 Space Control Register (CECTL) Field Values

Pin Information			Configuration	
Bit	**Field**	**Symbol Val**	**Value**	**Description**
31−28	WRSETUP	OF(value)	6	Write setup width. Number of clock cycles‡ of setup time for address (EA), chip enable (CE), and byte enables (BE) before write strobe falls. For asynchronous read accesses, this is also the setup time of AOE before ARE falls.
27−22	WRSTRB	OF(value)	12	Write strobe width. The width of write strobe (AWE) in clock cycles.+
21−20	WRHLD	OF(value)	1	Write hold width. Number of clock cycles‡ that address (EA) and byte strobes (BE) are held after write strobe rises. For asynchronous read accesses, this is also the hold time of AOE after ARE rising.
19−16	RDSETUP	OF(value)	6	Read setup width. Number of clock cycles‡ of setup time for address (EA), chip enable (CE), and byte enables (BE) before read strobe falls. For asynchronous read accesses, this is also the setup time of AOE before ARE falls.
15−14	TA	OF(value)	2	Minimum Turn-Around time. Turn-around time controls the minimum number of ECLKOUT cycles‡ between a read followed by a write (same or different CE spaces), or between reads from different CE spaces. Applies only to asynchronous memory types.
13−8	RDSTRB	OF(value)	12	Read strobe width. The width of read strobe (ARE) in clock cycles.
7−4	MTYPE	ASYNC32	2h	32-bit-wide asynchronous interface.
3	WRHLDMSB	OF(value)	0	Write hold width MSB is the most-significant bit of write hold.
2−0	RDHLD	OF(value)	1	Read hold width. Number of clock cycles‡ that address (EA) and byte strobes (BE) are held after read strobe rises. For asynchronous read accesses, this is also the hold time of AOE after ARE rising.

EMIF CE3 Space Control Register (CECTL) Field Values

Pin Information			Configuration	
Bit	**Field**	**Symbol Val**	**Value**	**Description**
31−28	WRSETUP	OF(value)	1	Write setup width. Number of clock cycles‡ of setup time for address (EA), chip enable (CE), and byte enables (BE) before write strobe falls. For asynchronous read accesses, this is also the setup time of AOE before ARE falls.
27−22	WRSTRB	OF(value)	3	Write strobe width. The width of write strobe (AWE) in clock cycles.+
21−20	WRHLD	OF(value)	2	Write hold width. Number of clock cycles‡ that address (EA) and byte strobes (BE) are held after write strobe rises. For asynchronous read accesses, this is also the hold time of AOE after ARE rising.
19−16	RDSETUP	OF(value)	4	Read setup width. Number of clock cycles‡ of setup time for address (EA), chip enable (CE), and byte enables (BE) before read strobe falls. For asynchronous read accesses, this is also the setup time of AOE before ARE falls.
15−14	TA	OF(value)	2	Minimum Turn-Around time. Turn-around time controls the minimum number of ECLKOUT cycles‡ between a read followed by a write (same or different CE spaces), or between reads from different CE spaces. Applies only to asynchronous memory types.
13−8	RDSTRB	OF(value)	12	Read strobe width. The width of read strobe (ARE) in clock cycles.
7−4	MTYPE	ASYNC32	2h	32-bit-wide asynchronous interface.
3	WRHLDMSB	OF(value)	0	Write hold width MSB is the most-significant bit of write hold.
2−0	RDHLD	OF(value)	1	Read hold width. Number of clock cycles‡ that address (EA) and byte strobes (BE) are held after read strobe rises. For asynchronous read accesses, this is also the hold time of AOE after ARE rising.

EMIF CE2 Space Control Register (CECTL) Field Values

Pin Information			Configuration	
Bit	**Field**	**Symbol Val**	**Value**	**Description**
31−7	Reserved		0	Reserved. The reserved bit location is always read as 0. A value written to this field has no effect.
6	SNCCLK		0	Synchronization clock selection bit.
5	RENEN	READ	1	Read enable mode. SADS/SRE signal acts as SRE signal. SRE goes low only for reads. No deselect cycle is issued. (used for FIFO interface).
4	CEEXT	ACTIVE	1	On read cycles, the CE signal will go active when SOE goes active and will stay active until SOE goes inactive. The SOE timing is controlled by SYNCRL. (used for synchronous FIFO reads with glue, where CE gates OE).
3−2	SYNCWL		0	Synchronous interface data write latency.
1−0	SYNCRL		1	Synchronous interface data read latency.

EMIF CE3 Space Control Register (CECTL) Field Values

Pin Information			Configuration	
Bit	**Field**	**Symbol Val**	**Value**	**Description**
31−7	Reserved		0	Reserved. The reserved bit location is always read as 0. A value written to this field has no effect.
6	SNCCLK		0	Synchronization clock selection bit.
5	RENEN	READ	1	Read enable mode. SADS/SRE signal acts as SRE signal. SRE goes low only for reads. No deselect cycle is issued. (used for FIFO interface).
4	CEEXT	ACTIVE	1	On read cycles, the CE signal will go active when SOE goes active and will stay active until SOE goes inactive. The SOE timing is controlled by SYNCRL. (used for synchronous FIFO reads with glue, where CE gates OE).
3−2	SYNCWL		0	Synchronous interface data write latency.
1−0	SYNCRL		1	Synchronous interface data read latency.

Parameters and descriptions taken from TI document SPRA543.pdf. Information for the FIFOs taken from SN74V293.pdf. Information for the EMIF of the DSK 6416 is taken from TMS320C6416.pdf.

Timing Parameters And Values SPRA543

EMIF—Input Timing Requirement Definitions

Timing Parameter	Definition	Value	Min	Max	Unit
tsu	Data setup time, read D before CLKOUT1 high	tsu	6.5	-	ns
th	Data hold time, read D after CLKOUT1 high	th	1	-	ns

EMIF—Output Timing Characteristics (Data, Address, Control)

Timing Parameter	Definition	Value	Min	Max	Unit
td	Output delay time, CLKOUT1 high to output signal valid	td	1.3	7.1	ns

EMIF—Other Timing Descriptions

Timing Parameter	Definition	Value	Min	Max	Unit
tskew	Output skew time, represents the possible skew between output control/data signals for a set of operating conditions (temperature and voltage). For 'C6000 devices, tskew =< 2ns.		-	2	ns
tcyc	Clock cycle time.	720 MHz	-	1.388889	ns

FIFO—Input Timing Requirement

Timing Parameter	Definition	Value	Min	Max	Unit
trp(m)	Minimum required read pulse width	720 MHz	-	0.694444	ns
twp(m)	Minimum required write pulse width	720 MHz	-	0.694444	ns
tsu(m)	Data setup time required by the FIFO for writes, write D before write enable (async) or CLK (sync) high	tds	1.5	-	ns
tih(m)	Data hold time required by FIFO for writes, write D after write enable (async) or CLK (sync) high	tdh	0.5	-	ns
toe(m)	Output enable time, /OE low to output valid	toe	2	4.5	ns
toez(m)	Output disable time, /OE high to output high impedance	tohz	2	4.5	ns
tens(m)	Enable setup time, time required from /OE, /REN, /WEN low to UNCK high	tens	1.5	-	ns
tenh(m)	Enable hold time, time required from /OE, /REN, /WEN low after UNCK high				
tckh	Clock high time	tclkh	2.5	-	ns
tckl	Clock low time	tclkl	2.5	-	ns
trc(m)	Length of the read cycle		-	6	ns
twc(m)	Length of the write cycle		-	6	ns

FIFO—Output Timing Characteristics

Timing Parameter	Definition	Value	Min	Max	Unit
tacc(m)	Access time, from RCLK or /RE to ED valid	ta	2	4.5	ns
toh(m)	Output hold time				

External Logic—Output Timing Characteristics

Timing Parameter	Definition	Value	Min	Max	Unit
tpl	Logic propagation, delay due to external logic.	GUESS	-	1	ns

Timing Constraints for FWFT Write Interface

Setup >= 1	
Setup + Strobe >= (tsu(m) + tskew) /tcyc	2.52
Setup + Strobe >= (tens(m) + tplmax + tskew)/tcyc	3.24
Strobe >= (tckl(m))/tcyc	1.8
Hold + Setup >= (tckh(m))/tcyc	1.8
Setup + Strobe + Hold >= (twc(m))/tcyc	4.32
Hold >= (tih(m) + tskew)/tcyc	1.8

Timing Constraints for FWFT Read Interface

Setup >= 1	
Setup + Strobe >= (tdmax + tplmax + toe(m) + tsu)/tcyc	13.752
Setup + Strobe >= (tens(m) + tplmax + tskew)/tcyc	3.24
Strobe >= (tckl(m))/tcyc	1.8
Hold + Setup >= (tckh(m))/tcyc	1.8
Setup + Strobe + Hold >= (trc(m))/tcyc	4.32
Hold + Setup + Strobe >= (tdmax + tacc(m) + tsu)/tcyc	9.432

Previous Senior Design Group (For Comparison)

WRSETUP	6
WRSTRB	12
WRHOLD	1
RDSETUP	4
RDSTRB	12
RDHOLD	1
TA	2

Values found based on calculations above:

Write Interface Values

Setup	1
Strobe	3
Hold	2
TA	-

Read Interface Values

Setup	1
Strobe	12
Hold	2
TA	-

EDMA Configuration

Reference parameters against the data structure in code or header file.

Send configuration:
option register:
- priority: medium
- element size: 32 bit
- 2D source: no
- source address update mode: increment
- 2D destination: no
- destination address update mode: fixed
- transfer complete interrupt: yes
- transfer complete code: TCC_SEND (#defined in source)
- transfer complete code extra bits: 00
- alternate transfer complete (ATC) interrupt: no
- ATC code: don't care
- peripheral device source mode: disable
- peripheral device destination mode: disable
- link: no
- frame synchronization: yes

Source address register: pingpong[i]
Count register:
frame count: 0 (1 frame)
element count: PINGPONG_SIZE (number of elements per pingpong buffer)
Destination address register: 0xB0000000 = EMIFA CE3
index register: default (don't care)
reload register: default (don't care)

Receive configuration:
option register:
- priority: medium
- element size: 32 bit
- 2D source: no
- source address update mode:fixed
- 2D destination: no
- destination address update mode: increment
- transfer complete interrupt: yes
- transfer complete code: TCC_RECV (#defined in source)
- transfer complete code extra bits: 00
- alternate transfer complete (ATC) interrupt: no
- ATC code: don't care
- peripheral device source mode: disable
- peripheral device destination mode: disable
- link: no
- frame synchronization: yes

Source address register: 0xA0000000 = EMIFA CE2
Count register:
frame count: 0 (1 frame)
element count: PINGPONG_SIZE (number of elements per pingpong buffer)
Destination address register: pingpong[i]

index register: default (don't care)
reload register: default (don't care)

Timer Configuration

Timer 0 and Timer 1
Timer clock frequency = 45 MHz = CPU frequency / (8*2)
Timer CTL Register:
input inversion: don't care if clock source is not external
clock source: internal (CPU frequency / 8)
clock or pulse: clock (50% duty cycle)
hold: no (counter is allowed to count)
go: no (don't care if hold = no)
pulse width: don't care if 50% duty cycle
data out: TOUT
invert output: no
TOUT function: timer (note: timer output can be disabled here)

Daughter Card Control Register

Bit	Name	R/W	Description	Value
7	DC_DET	R	Daughter card detect	0
6	0	R	Always 0	0
5	DC_STAT1	R	Daughter card status	0
4	DC_STAT0	R	Daughter card status	0
3	DC_RST	R/W	Daughter card reset	0
2	0	R	Always 0	0
1	DC_CNTL1	R/W	Daughter card control	0
0	DC_CNTL0	R/W	Daughter card control, controls the partial reset of time reversal daughter card	1

Appendix B – Code

Appendix C – Schematics

Appendix D – PCB Files